**Using the C language to solve these linked list assignments:**

1. Memory Management Control: C provides low-level control over memory usage via pointers. This is particularly useful for data structures like linked lists, where understanding and managing memory allocation and deallocation is crucial. Working with pointers directly helps students understand the memory model more deeply.

2. Pointer Manipulation Proficiency: Linked lists inherently rely on pointers to connect their elements. C's direct manipulation of pointers offers a hands-on experience to manipulate nodes, links, and data structures, enhancing the learner's understanding of how data is interconnected in memory.

3. Foundation for Understanding Complex Systems: Many underlying systems and algorithms in software engineering and computer science, such as operating systems and file systems, are implemented in C. Proficiency in C and linked lists gives students a better understanding of these systems.

4. Performance: C is a compiled language known for its performance and speed. For operations involving large data and linked structures, C can handle tasks efficiently without the overhead of automatic memory management and other abstractions found in higher-level languages.

5. Industry Relevance: Despite the age of the language, C remains widely used in industry, particularly in systems programming, embedded systems, and scenarios where hardware interaction is necessary. Learning linked lists in C not only provides educational benefits but also practical skills applicable in many technical careers.

Linked List Assignments in C

1. Implement a Singly Linked List: Create the basic structure of a singly linked list and implement programs for insertion, deletion, and display of the list.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct List{

    int data;

    struct List \*next;

} List;

//INSERTION

List\* insert\_at\_beginning(List\* head, int data) {

    //This function aims to insert a node in the first place

    if (head == NULL){//If the linked list is empty

        return head;

    }

    // Create a new node by dynamically allocating a memory space to it.

    List\* new\_node = (List\*)malloc(sizeof(List));

    new\_node->data = data;//adding input data to the node

    new\_node->next = head;// making the node point to the head

    return new\_node;

}

List \*insert\_at\_last(List\* head, int data){

    // This function aims to insert a node at the end of the list

    // Create a new node by dynamically allocating a memory space to it.

    List\* new\_node = (List\*)malloc(sizeof(List));

    new\_node->data = data;

    new\_node->next = NULL;

    List \*temp = head;

    while (temp->next != NULL){

        //traversing to the last point of the linked list

        temp = temp->next;

    }

    temp->next=new\_node;// The temporary node pointing to the new node which points to the NULL pointer

    return head;

}

// DELETION

List\* delete\_first(List\*head){//The function tries to delete the first node of the linked list

    if (head == NULL){// If the linked list is empty

        return NULL;

    }

    //Initializing a temporary pointer which points to the head

    List \*temp = head;

    //Moving the head pointer to the next node

    head = head -> next;

    free(temp); //Freeing the temporary pointer

    return head;

}

List \*delete\_last(List\* head){

    if (head == NULL){// If the linked list is empty

        return NULL;

    }

    if (head->next == NULL){// If the linked list has only one node in it

        free(head);// Freeing  or deleting the node

        return NULL;

    }

    List\* temp = head; // initializing the temporary pointer to the head pointer

    List \*prev = NULL; // Initializing the previous pointer to the null pointer

    while(temp -> next != NULL){

        // Traversing the temporary pointer to the last node node

        prev = temp;

        temp = temp -> next;

    }

    prev->next = NULL;

    free(temp);

    return head;

}

// PRINTING THE LIST

void display(struct List\* node1){

// This function aims to print all the nodes in the linked list

    List \*temp = node1;

    temp = node1;

    // Initializing the temporary pointer to the head of the linked list

    while(temp != NULL){

        printf("%d -> ", temp->data);

        temp = temp -> next;

    }

    printf("NULL \n"); // Printing the NULL pointer

}

int main(void){

    List \*node1 = (List\*)malloc(sizeof(struct List));

    List \*node2 = (List\*)malloc(sizeof(struct List));

    List \*node3 = (List\*)malloc(sizeof(struct List));

    node1 -> data = 132;

    node2 -> data = 134;

    node3 -> data = 145;

    node1 -> next = node2;

    node2 -> next = node3;

    node3 -> next = NULL;

    node1 = insert\_at\_beginning(node1, 345);

    node3 = insert\_at\_last(node3, 324);

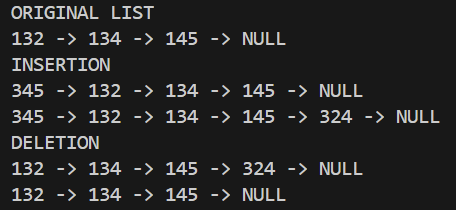
    node1 = delete\_first(node1);

    node3 = delete\_last(node3);

    display(node1);

}

**OUTPUT:**



1. Find the Middle Element: Implement an algorithm to find the middle element of the linked list in a single pass.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct List{

    int data;

    struct List \*next;

} List;

List \*middle\_element(List\* head){

    // This function tries to find the middle element in a given

    if (head == NULL){// If the linked is empty

        return NULL;

    }

    if (head->next == NULL){ // If the linked list has only one node in it

        return head;

    }

    //Using the tortoise hare method

    List \*slow = head; // Initializing the slow pointer to head

    List \*fast = head; // initializing the fast pointer to head

    while (fast != NULL && fast->next != NULL){ // Until fast reaches the NUll pointer and reaches the last pointer

        // Slow will traverse the linked list slower compared to fast pointer

        slow = slow->next;

        fast = fast->next->next;

    }

    // Returns the slow pointer which will be in the middle of the list.

    return slow;

}

void display(struct List\* node1){

    //Thsi function aims to print all the nodes in the linked list

    List \*temp = node1;

    temp = node1;

    // Initializing the temporary pointer to the head of the linked list

    while(temp != NULL){

        printf("%d -> ", temp->data);

        temp = temp -> next;

    }

    printf("NULL \n"); // Printing the NULL pointer

}

int main(void){

    List \*node1 = (List\*)malloc(sizeof(struct List));

    List \*node2 = (List\*)malloc(sizeof(struct List));

    List \*node3 = (List\*)malloc(sizeof(struct List));

    node1 -> data = 132;

    node2 -> data = 134;

    node3 -> data = 145;

    node1 -> next = node2;

    node2 -> next = node3;

    node3 -> next = NULL;

    printf("LINKED LIST \n");

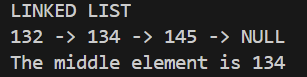
    display(node1);

    List \*middle = middle\_element(node1);

    printf("The middle element is %d", middle->data);

}

**OUTPUT:**



1. Detect a Loop: Write a program that detects if a loop exists in a linked list. If a loop is found, your program should be able to identify the starting point of the loop.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct List{

    int data;

    struct List \*next;

} List;

List \*detect\_loop(List\* head){

    // This function tries to detect a loop in the linked list

    if (head == NULL || head->next == NULL){ // If the linked list is empty or has only one node

        return NULL;

    }

    List \*sloop = head;// Initializing the sloop(start of loop) pointer to head

    List \*eloop = head;// Initializing the eloop (end of the loop) pointer to head

    while (eloop != NULL && eloop->next != NULL){// Until eloop pointer reaches the last node or null pointer

        sloop = sloop -> next;

        eloop = eloop->next->next;

        // Detection of a loop

        if (sloop == eloop){

            break;

        }

    }

    if (eloop == NULL || eloop->next == NULL){ // Handles the case where the null pointer pointes to another null pointer

        return NULL;

    }

    // Finding the loop in the linked list

    sloop = head;

    while(sloop != eloop){

        sloop = sloop -> next;

        eloop = eloop -> next;

    }

    // Returns the starting point of the loop

    return sloop;

}

void display\_loop(struct List\* head){

    // This function tries to display the start and the end of the loop

    List \*loop = detect\_loop(head);

    List \*temp = head;

    while (temp !=  NULL){

        printf("%d -> ", temp->data);

        if (temp == loop){

            printf("Loop starts here");

            break;

        }

        temp = temp -> next;

    }

    if (loop == NULL){

        // If the loop is not detected

        printf("NULL \n");

    } else {

        // If the loop is detected

        printf("-> %d", temp->data);

    }

}

void display(struct List\* node1){

    //Thsi function aims to print all the nodes in the linked list

    List \*temp = node1;

    temp = node1;

    // Initializing the temporary pointer to the head of the linked list

    while(temp != NULL){

        printf("%d -> ", temp->data);

        temp = temp -> next;

    }

    printf("NULL \n"); // Printing the NULL pointer

}

int main(){

    List \*node1 = (List\*)malloc(sizeof(List));

    List \*node2 = (List\*)malloc(sizeof(List));

    List \*node3 = (List\*)malloc(sizeof(List));

    List \*node4 = (List\*)malloc(sizeof(List));

    List \*node5 = (List\*)malloc(sizeof(List));

    node1 -> data = 132;

    node2 -> data = 134;

    node3 -> data = 145;

    node4 -> data = 132;

    node5 -> data = 293;

    node1 -> next = node2;

    node2 -> next = node3;

    node3 -> next = node4;

    node4 -> next = node5;

    node5 -> next = NULL;

    printf("LINKED LIST \n");

    display(node1);

    node4 -> next = node1;

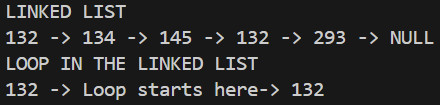
    printf("LOOP IN THE LINKED LIST \n");

    List \*loop = detect\_loop(node1);

    display\_loop(loop);

}

**OUTPUT:**



1. Reverse a Linked List: Write a program to reverse a singly linked list without using additional lists.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct List{

    int data;

    struct List \*next;

} List;

List \*reverse\_list(List\* head){

    //Tsi function tries to reverse a linked list using the concept of recursion

    if (head == NULL || head->next== NULL){// If the linked list has no nodes or has only one node

        return head;

    }

    List \*rest = reverse\_list(head -> next);

    head -> next -> next = head;//swapping of thre head pointer with other nodes

    head -> next = NULL;// The recursion ends as soon head points to the null pointer

    return rest;

}

void display(struct List\* head){

    // Prints the reversed linked list

    List \*temp = head;

    while (temp !=  NULL){

        printf("%d -> ", temp->data);

        temp = temp -> next;

    }

    printf("NULL \n");

}

int main(){

    List \*node1 = (List\*)malloc(sizeof(List));

    List \*node2 = (List\*)malloc(sizeof(List));

    List \*node3 = (List\*)malloc(sizeof(List));

    List \*node4 = (List\*)malloc(sizeof(List));

    List \*node5 = (List\*)malloc(sizeof(List));

    node1 -> data = 132;

    node2 -> data = 134;

    node3 -> data = 145;

    node4 -> data = 132;

    node5 -> data = 293;

    node1 -> next = node2;

    node2 -> next = node3;

    node3 -> next = node4;

    node4 -> next = node5;

    node5 -> next = NULL;

    display(node1);

    List \*reverse = reverse\_list(node1);

    printf("REVERSED LINKED LIST \n");

    display(reverse);

    free(node1);

    free(node2);

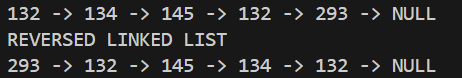
    free(node3);

    free(node4);

    free(node5);

}

**OUTPUT:**



1. Remove Duplicates from a Sorted List: Develop a program that removes duplicate elements from a sorted linked list.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct List{

    int data;

    struct List \*next;

} List;

List\* remove\_dups(List\* head, int data){

    // Using an iterator pattern to traverse and modify the list

    List\* temp = head; //Initializing the temporary pointer to the head

    List\* prev = NULL; // Initializing the prev pointer to NULL

    while (temp != NULL) { // Until temp becomes NULL

        if (temp->data == data) {

            // Found a node to remove

            if (prev == NULL) {

                // Removing the head node

                List\* current = temp;

                temp = temp->next; // Move to the next node

                head = temp; // Update head

                free(current); // Free the removed node

            } else {

                // Removing a node in the middle or end

                prev->next = temp->next;

                free(temp);

                temp = prev->next; // Move temp forward

            }

        } else {

            // Move both pointers forward when data doesn't match

            prev = temp;

            temp = temp->next;

        }

    }

    return head;

}

void display(struct List\* head){

    // Iterator pattern for displaying the linked list

    List \*temp = head;

    while (temp != NULL) {

        printf("%d -> ", temp->data);

        temp = temp->next;

    }

    printf("NULL \n");

}

int main() {

    // Creating the list nodes

    List \*node1 = (List\*)malloc(sizeof(List));

    List \*node2 = (List\*)malloc(sizeof(List));

    List \*node3 = (List\*)malloc(sizeof(List));

    List \*node4 = (List\*)malloc(sizeof(List));

    List \*node5 = (List\*)malloc(sizeof(List));

    // Initializing the list with values

    node1->data = 132;

    node2->data = 132;

    node3->data = 145;

    node4->data = 132;

    node5->data = 293;

    // Linking the nodes

    node1->next = node2;

    node2->next = node3;

    node3->next = node4;

    node4->next = node5;

    node5->next = NULL;

    // Displaying the original list

    display(node1);

    printf("Removing duplicates of 132\n");

    // Removing duplicates and displaying the modified list

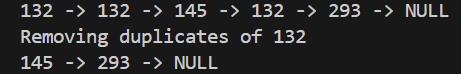
    List \*new\_list = remove\_dups(node1, 132);

    display(new\_list);

    return 0;

}

**OUPUT:**



1. Add Two Numbers Represented by Linked Lists: Given two numbers represented by two linked lists, where each node contains a single digit, write a program that adds the two numbers and returns the sum as a linked list.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Link{

    int tens;

    struct Link \*next;

} Link;

typedef struct Link2{

    int tens1;

    struct Link2 \*next;

} Link2;

Link\* createNewnode(int tens){

    // attempts to create a newnode and add it to the exsisting linked list

    Link\* newnode = (Link\*)malloc(sizeof(Link));

    newnode -> tens = tens;

    newnode -> next = NULL;

    return newnode;

}

Link2\* createNewnode2(int tens1){

    // attempts to create a newnode and add it to the exsisting linked list

    Link2\* newnode = (Link2\*)malloc(sizeof(Link2));

    newnode -> tens1 = tens1;

    newnode -> next = NULL;

    return newnode;

}

Link\* sum\_of\_lists(Link \*head1, Link2 \*head2){

    // this function aims to find the sum of the two linked lists given

    if (head1 == NULL && head2 == NULL){

        return NULL;

    }

    //Create two temporary lists to traverse through the lists

    Link \*temp1 = head1;

    Link2 \*temp2 = head2;

    while (temp1 != NULL && temp2 != NULL){

        // adding the contetns of the list

        temp2 ->tens1 += temp1 -> tens;

        // traversing the list

        temp1 = temp1 -> next;

        temp2 = temp2 -> next;

    }

    //returns the head of the new list

    return head2;

}

void display(struct Link\* head1, Link2\* head2){

    // aims to display the list or print the list

    Link2 \*loop = sum\_of\_lists(head1, head2);

    Link2 \*temp = loop;

    while (temp !=  NULL){

        printf("(%d) -> ", temp->tens1);

        temp = temp -> next;

    }

    printf("NULL \n");

}

void display\_normal(Link\* head){

    Link\* temp = head;

    while (temp !=  NULL){

        printf("(%d) -> ", temp->tens);

        temp = temp -> next;

    }

    printf("NULL \n");

}

void display\_normal2(Link2\* head){

    Link2\* temp = head;

    while (temp !=  NULL){

        printf("(%d) -> ", temp->tens1);

        temp = temp -> next;

    }

    printf("NULL \n");

}

int main(void) {

    Link\* node1 = createNewnode(5); // 5

    Link\* node2 = createNewnode(7); // 7

    node1->next = node2;

    Link2\* n1 = createNewnode2(2); // 2

    Link2\* n2 = createNewnode2(1); // 1

    n1->next = n2;

    // display the sum of the two lists

    display\_normal(node1);

    display\_normal2(n1);

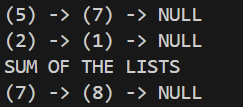
    printf("SUM OF THE LISTS\n");

    display(node1, n1);

    return 0;

}

**OUTPUT:**



**----------------------------------- Good luck ----------------------------------**